Status of natural language generation and its implementation using register vector grammar /

Marie Bettinger Wilde
Lehigh University

STATUS OF NATURAL LANGUAGE GENERATION AND ITS IMPLEMENTATION USING REGISTER VECTOR GRAMMAR

by

Marie Bettinger Wilde

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

Computer Science and Electrical Engineering

Lehigh University

1986

Certificate of Approval

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

9/29/86

(date)

[illegible]

Professor in Charge

[illegible]

Chairman of Department

Acknowledgements

I gratefully acknowledge the invaluable assistance of Glenn Blank, an invaluable source of interesting, challenging ideas, creative ways to implement them, and plenty of time to discuss them, a mentor in every sense of the word. I would also like to thank my mentor's mentor, Art Kunst, for his input.

Finally, I am grateful for the understanding of my family and the tolerance of my co-workers as I pursued this research often to the neglect of all else.

TABLE OF CONTENTS

Abstract

This thesis explores one area of artificial intelligence research: text generation. The topic is developed in the following way:

a. a brief review of the research, the past, present and future of natural language generation;

b. an explanation of Register Vector Grammar (RVG), a formalism for parsing and generating; and

c. a methodology for using RVG to generate text that is coherent with respect to syntactic, semantic and referential constraints.

Chapter 1 - Natural Language Generation Research

A. Introduction

The process by which a child learns to speak a language is nothing short of miraculous. Having personally observed two human beings progress in just a few years from uttering little more than a grunt to parsing and generating highly sophisticated discourse and having compared their capability to that of the computer under years of tutorage by computational linguists, I concede that this area of artificial intelligence has yet a long way to go.

Even on a child's level of speech, the manufacture of sentences is quite technical. McDonald attests to the complexity of the process:

> "Even if we ... look just at the "linguistic" part of the process - selecting words and constructions, applying grammatical rules, and producing the words in sequence - it is clear that very sophisticated rules are being followed. Somehow we select one lexical/syntactic combination from the many possible alternatives, managing to attend simultaneously to the potentials of the different constructions, our multiple goals, and the constraints arbitrarily imposed by our grammar. We follow conventions of direct utility only to our audiences and activaly maintain elaborate coherency relations across large stretches of discourse". [McDonald 80]

To expect the computer to achieve this level of competency in a short period of time is unrealistic, but research has been chipping away at the problems, and a number of text generation systems are in use today. Moreover, as the need for the creation of quality text continues to grow, we will undoubtedly see research in new areas as well as further study of those aspects of generation which are now under active investigation.

B. The Demand

Despite widespread use of the computer, the user community is unwilling to live with the inconveniences of the machine, most notably, the lack of a refined man-machine interface. A number of factors are contributing to the demand:

1. As the number of untrained users grows, so does the need to equip computers with the ability to communicate in natural language. Almost any system is greatly constricted in its use because it cannot communicate with the amateur.

2. Systems are becoming more complex. It is no longer possible to predict during programming every possible output which will be

required in such areas as process control and expert systems. These and other systems need to be able to generate original text.

3. Systems should be capable of generating output of many different forms, status reports, rationales for action taken, alternative data paths, etc.

4. The techniques of prototyping in system analysis and design use a fourth generation language to construct a model for user review. Because of the many changes necessary during the process, a natural language interface would greatly improve this task.

5. Text generation used in query languages needs to be improved so that resources in a data base can be fully tapped.

6. Communication in a variety of robotics environments would be enhanced by capabilities to generate meaningful discourse.

7. Finally, a long-range goal for natural language generation is well-written, quality, multi-paragraph text for a variety of linguistic purposes including machine translation, synopsis, and reporting.

These demands for natural language in a

computer environment, as well as many others which will undoubtedly surface in the next decade, have established text generation as one of the most vital areas of AI research.

C. Text Generation Systems

A review of some of the important text generation systems follows.

1. The Kafka Generator [Mauldin 84]

1.1 Purposes and Design

XCALIBUR is a natural language interface used to extract data from and answer queries to expert systems. The Kafka generator is implemented by XCALIBUR to produce a single sentence.

The Kafka generator performs the following functions as part of the XCALIBUR system:

a) formulates replies to user queries to expert systems,

b) clarifies poor input from the user to the system, and

c) formulates system questions to the user.

Kafka uses an interpreter written in Franz Lisp to convert input from XCALIBUR (which is in a case frame format) or from a relational database into conceptional dependency graphs and then into English, implementing both semantic and syntactic processing. Kafka receives a case frame from XCALIBUR and uses transformational rules to produce new case frames, with lexical items as output in the process. Recursion is used to instantiate subcases of the parent node of the CD graph.

1.2 Generation Phases

Generation by Kafka is accomplished by a number of modules, each of which performs a sub-process. First, preprocessing simplifies case frame input, stripping unnecessary syntactic information and clarifying complex frames. Then, as required for queries, interrogative form is changed to declarative with placeholders for information in the reply. The database then fills in these blanks, and a conceptual dependency graph is constructed. Next, a verb is selected for each CD primitive, then adjectives and nouns are added. Finally the leaves of the tree are printed.

2. Knowledge Delivery System [Mann & Moore 81]

2.1 Purpose and Design

The purpose of KDS is to produce well-written multiparagraph text. One of the major problems in computer generation of text is that there is no concrete theory of writing to explain how people create text. In constructing KDS, Mann and Moore set out to explore the art of writing and, in so doing, add to the understanding of the process of writing, as well as the knowledge of how to simulate it by computer.

Multiparagraph text generation previous to KDS was designed to use a Partitioning paradigm. These systems process input via large information structures which must be broken down into smaller pieces suitable for single sentences. Sentences are formed by traversing the data structure. Several problems result in this process: information pertinent for one sentence may not be adjacent in the data structure; there may be useless pieces of information after extracting sentences; and the difficulty of determining good sentence boundaries in the remaining structure is problematic.

These deficiences caused Mann and Moore to reject Partitioning and adopt a Fragment-and-Compose Paradigm, which, rather than systematically biting chunks of information and forming sentences in order from the data structure, instead fragments the structure into propositional parts and then composes meaningful sentences and meaningful paragraphs.

2.2 Generation Modules

There are five modules involved in the process:

a. Fragmenter, which divides the input into small units;

b. Problem-Solver, which determines style and organization of the text;

c. Knowledge-Filter, which removes redundant information;

d. Hill-Climber, which repeatedly improves the quality of the protosentences, combines the clauses, then compares options;

e. Surface-Sentence-Maker, which formulates final sentences using a context free grammar and semantic rules.

An important contribution of KDS research is the concept of goal pursuit: the program and the system are designed to seek the best way of saying something, given the knowledge to be communicated. This was the first implementation of revision in machine generated text, a topic which is receiving much attention in today's research (to be discussed later).

3. MUMBLE [McDonald 80]

3.1 Purpose and Design

McDonald's system generates immediate speech, the conversational type in which the party does not know what is to be said in subsequent utterances. The system operates in an on-line framework, like conversation, generating from "messages" that are passed to the linguistic component. There are two transducers, one forming a knowledge representation from the message, and the second producing text from the data structure. These transducers answer to the Controller.

3.2 Analysis of the Generation

At any given time, the state of the linguistic component can be viewed as a four-dimensional array consisting of the following information:

a. the name of the controller process subroutine (Process-node, Process-slot, and Dispatch),

b. the value of the three controller variables (current-node, current-slot, current-contents),

c. the value of the grammar variables,

d. the record of the discourse history.

4. Other Text Generators

Having reviewed three interesting systems for text generation, I mention also the following important systems currently in use:

a. TEXT [McKeown 82] - uses schemas to define text regions that satisfy predicates

b. PROTEUS [Davey 79] - uses semantic nets to describe tic-tac-toe games in multiparagraph text

c. KAMP [Appelt 81] - generates single sentence output

d. RACTER [reviewed by Kenner 86]- engages in somewhat meaningless babbling with a user

D. Analysis of a Generator

On the lowest level of generation is a random syntactic generator. One step above canned text, the program is supplied with a grammar of rules which are enforced at random. If a particular rule has a lexical category attached to it, a word of the sentence is generated from within the program. In the next refinement, the word is selected at random from a lexicon. The rules of the grammar have provisions for wh-questions, yes-no questions, clause embedding, participles, active and passive voice.

In the gray area between syntax and semantics, various morphology and agreement rules must be enforced: agreement of the verb with its subject, case of pronouns, plural forms, participles.

On a higher level, the generator implements semantics. Predicative semantics accumulates semantic infomation from the words generated and places constraints on subsequent words. The predicate structure thus formed assures that arguments generated from the lexicon satisfy the semantic requirements for the particular text being generated. Predicative semantics operates in much the same way as the semantic theory of psychology [Smith et al 74], which argues that people store semantic features in their mental lexicon and use these when generating sentences.

Referential semantics increases the depth of semantic processing by making the semantic information part of a permanent database. This referent list can be maintained, updating with new entries and adding qualities of old ones almost indefinitely, therefore broadening the scope of text for which the system can assure semantic agreement. Grosz et al [83] consider this a vital aspect of generation and implement it conceptually as focusing (discussed below).

E. Issues and Problems

As mentioned above, a major problem in creating text by machine is that, despite a large body of knowledge in linguistics, we have no concrete theory detailing how people go about creating text. The heuristics which are part of writing skills and the psycholinguistics of the process itself have been the object of intense research for over a decade, but it will be some time before experts agree on the issues below and the problems are solved to an extent where sophisticated text generation is accomplished.

1. The Grammar

The systems discussed above, and most others, are operating with a very limited subset of a grammar. This is due, in part, to the fact that grammars have come from the ranks of linguists, who do not usually impose upon a grammar the rigor necessary for computational processing, parsing or generation. Some researchers feel that a sophisticated and complete grammar requires the work of competent linguists and cannot be developed by computer scientists[Mann 82]. Systems operating with only a fragment of a grammar cannot hope to generate text which approximates human-produced material, and, moreover, risk creating subtle misconceptions and erroneous images in the minds of readers.

On a larger scale, a restrictive grammar limits the applicability of the system; the program can then generate only a very predictably small variety of text.

The formalism used to describe the grammar is also at issue. A recent project by Derr and McKeown [86] uses Definite Clause Grammar, whose rules are composed of first order predicate logic clauses. The DCG formalism solves a number of sticky generation problems in the following ways:

a. nonterminals have arguments which can hold the string being analyzed

b. the parse tree representation can be built and passed easily

c. agreement and morphology information can be maintained easily

d. encoding is facilitated by the ability to carry extra conditions or flags with the grammar rules.

Because DCG lends itself well to linguistic processing, it is likely to be an important subject of future research.

2. The Input Representation

The way in which the knowledge required by the generator is presented to it is critical in the effectiveness of the text produced. This knowledge representation of the deep structure varies widely with the system, running the gamut from predicate calculus to semantic nets. Many weaknesses in a generator result from the deficiences in the input representation, for example, the lack of concrete symbols to signify abstractions which may be present, such as possibility or likelihood. Future research will need to dramatically improve the knowledge representation to solve these kinds of problems.

3. Discourse Coherence

When correct syntax and semantics are achieved, text is successfully produced. But there are many ways by which one can judge the quality of that text and there is general agreement that systems must aim for better-written, more polished, smoother text with variety of expression, the kind created by people. The natural language generators of the future must do more than answer queries; they must write manuals and generate reports, translate effectively and completely, and perform the entire gamut of human writing functions, except perhaps on the most creative level.

The dimension added to syntax and semantics to elevate text to the level of discourse is pragmatics. Rosenchien [81] summarizes discourse theory as the process of studying "how linguitic events are interpreted and initiated for pragmatic effect". Discourse, then, can be seen as a goal-pursuing activity when a person engages in it, since words are chosen with the belief that one will accomplish the purpose one has in mind.

Grosz et al [83] identify two types of discourse coherence: global, which interconnects large segments of discourse to yield the general purpose of the text, and local, by which sentences contribute to the larger segments. Text generation needs to consider these separately.

There are a number of factors which affect local discourse coherence. A major consideration is definite reference, including deixis and anaphora, to be discussed below. This referencing brings cohesion to the text, as does ellipsis and coreferences (use of the same item twice).

In addition, other linguistic constructions need to be correctly and frequently applied, as they contributed continuity to text. These include the following connectives:

a. subordinate and coordinate conjunctions, such as "while", "because", and "but" ;

b. adverbials, such as "therefore", "obviously", and "accordingly" ;

c. sentential adjuncts , such as "first of all", "for example", and "that is" .

By connecting two segments of discourse explicitly, these constructs force increased understanding of the text.

Some of the theoretical aspects of discourse coherence which are being studied have interesting ramifications both linguistically and computationally. The given/new strategy, for example, deals with that aspect of discourse which purports that there is "shared knowledge," a set of propositions which the speaker and listener believe they have in common [Clark & Marshall 81]. Certainly this theory needs to be extended into generation. Another important theoretical issue is focusing, discussed below.

4. Focus and Definite Reference

Definite reference can be classified in various ways. Two important types are deixis and anaphora. Deixis concerns the use of declaratives such as "this" and "that". Anaphora considers all other cases of pronouns and definite description. Until recently, the interpretation of an anaphoric expression relied completely upon determining and analyzing it antecedent. Current research exposes this totally syntactic interpretation as insufficient to handle semantic and pragmatic factors. In language generation, a more comprehensive understanding of referents is vital, one which considers syntactic, semantic, and discourse constraints (see Sidner [81], Linde [79], and Grosz et al [83]).

The concept of focus provides a first step toward a solution to the problems of anaphora in generation. The focusing process affects local coherence, in the following way: the speaker, at any time in a discourse, focuses attention on a certain item in a context space, and indicates what that item is by his language; the listener must follow the speaker, choosing a new focus as the speaker introduces one, and establishing its validity from subsequent rhetoric. In machine generation, a semantic process must be conducted as foci change and are updated. This presents a number of problems, particularly when there are multiple foci, as in the following:

> My daughters, Lisa and Genny, are as different as night and day. They each have many talents, but it is obvious they will use them in different ways. Lisa is very creative and literate. She will undoubtedly perform in the theatre or critique it, whereas Genny will design it or build it, since she is an analytical and mechanical thinker. Yet it's true that I think they are both delightful.

The extensive use of anaphora and constant shifting of focus, while natural in discourse, will be demanding for computer generation.

5. Revision in Written Text

Discourse coherence in written text will be subject to more stringent demands, since the text is on a much higher level than single sentences or even multiple sentences in dialogue. The most recent thinking in this area is the rejection of a single-pass generator in deference to a more natural approach imitating the way people write, that is, with repeated revision. This introduces multiple opportunities for recognizing coherence difficulties on different levels and correcting them in a multi-phase approach. The KDS system uses a Hill-Climber module to generate with revision. The Penman system [Mann 83] represents the most recent technology in this theory, using a wider variety of methods of refinement at each stage of revision, such as paring down sentence length and limiting clause embedding, two traditional methods of revision in writing.

6. Text Organization

The effectiveness of generated text is strongly affected by the way in which it was planned. One facet of this planning is text organization: the structure of the text is clearly composed of well-defined parts, all of which are combined to form a unit.

Mann's Rhetorical Structure Theory [Mann 84] is one of the few studies which specifically explores this aspect of text planning. RST research has divided the study of text organization into descriptive theory, used to study naturally occuring text, and constructive theory, for the process of machine generated text. Mann uses goal-pursuit and a number of schemas as the means to accomplish quality text generation.

There will likely be other theories of text organization in the future, striving for Mann's characteristics:

a. comprehensiveness: usefulness for all types of text,

b. scale insensitivity: usefulness for any size text,

c. functionality: success in achieving the intended effect of the writer,

d. definiteness: adaptability to formalization,

e. generativity: applicability to both descriptive and constructive text.

Chapter 2 - An RVG Natural Language Generator

A. Introduction

Register Vector Grammar is a natural language processing system which represents syntactic and semantic content by using ternary feature vectors. Currently in place at Lehigh University is an RVG parser developed by Glenn Blank and written in Turbo Pascal, and this generator is an extension of that natural language system.

This chapter will describe the syntactic part of the generator RVGGEN and suggest directions for semantic processing via RVG. Extensive system documentation is available for RVG and no attempt will be made to reiterate that information.

B. Foundations of RVG

As noted above, RVG processes natural language using ternary vectors. These vectors, implemented in Pascal using sets, are ordered n-tuples of a fixed length, each element of which takes on a value of 0,1, or 2. Position in the vector is a placeholder for a certain syntactic or semantic characteristic; the value which stands in the placeholder represents, in general, absence (0), presence (1), or neutrality (2).

Ternary vectors are involved in both syntactic and semantic processing. The Current Syntactic State Register (CSSR) is a ternary vector which controls the order of syntactic productions; the Current Predication State Register (CPSR) controls generation of entries which in turn are composed of ternary vectors representing semantic properties.

C. Nature and Purpose

The (current) purpose of RVGGEN is to generate single sentence output using Register Vector Grammar and a lexicon. Syntactic generation proceeds as follows: An initial state value is placed in the CSSR. Subsequent iteration of the following process occurs until the final state is accepted: a production rule from the grammar whose condition vector matches the CSSR is applied, the state of the CSSR is advanced based on the result vector of the production rule.

The generator currently operates in one of two modes:

Brief - prints a single sentence

Trace - follows the generation of the sentence, printing CSSR values as they change.

D. Design

The program is written in Turbo Pascal using various input files. All but one of these files are converted from text to record format by a utility program, also in Pascal.

The main module is procedure Generate as follows:

```
Initialize variables and registers
Repeat
    form a linked list of all productions whose condition vector matches the CSSR
    choose one of these at random (to be discussed below)
    if it is a lexical production, print a random item of the appropriate lexical category
    change the CSSR according to the result vector of the production
Until the final state is reached.
```

It should be noted that at any given point in the generaton process there may be many, one, or no productions which may apply. If none are possible as a next state, the generation process fails. If only one applies, that step is taken. In most instances, the linked list SynQ will contain a number of possible productions whose condition vectors match the CSSR. In this case, a weight for each production (from an input file) is compared to a random number: if the weight exceeds the random number, the production succeeds; otherwise the next potential production is tested. If no production succeeds, a function chooses the highest weighted production in SynQ.

The weights file is sensitive to the frequency of English constructions. Thus when the syntactic state is conducive to a number of different possible productions, the determination will be made based on its likelihood in discourse. For example, the generation of a pronoun will occur with a higher frequency than a definite description.

E. Input

RVGGEN requires the following input files:

1. SynFile - This is a file of records converted by a utility program from a text file. Each record in the file consists of a grammatical label; two ternary vectors, a condition and a result; a register indicating if the production is a lexical one, one which requires morphological processing, or neither; a register marking that passage to a new clause level is indicated

by this production; a field to store the number of non-lexical productions which can precede this production; for non-lexical productions, the number of contraints; an array of indices to productions which can precede this one.

2. LexFile - Also converted from text to record format by the preprocessor, each lexical record consists of the alphabetic representation of the lexeme, an array of indices to lexical productions of the lexeme's category, and the number of grammatical categories to which this lexeme belongs.

3. MaxVecFile - This is a text file containing the length of the ternary vectors.

4. WeightFile - As explained above, this is a text file assigning a weight to each production rule. In a random generator, adjusting these weights will determine when one suitable natural language construct will be chosen over another.

5. LookupFile - This is a file of records produced in processing the lexicon file by the converting utility. The text file for the lexicon contains morphological forms for lexemes; this information is stored in LookupFile.

F. Output

Currently all output is interactive. The program prompts for a mode of operation as noted above. Additionally, the user is asked for the number of sentences to be generated. If Brief mode is selected, output consists of the number of sentences requested; if Trace mode is selected, the progression through the generation process is traced, including state changes, productions chosen by matching, and resultant effect upon the CSSR.

G. Future Directions

1. Predicative Semantics

1.1 Introduction

RVGGEN will next implement predicative semantics. Semantic information will be a part of the lexicon in the form of ternary semantic vectors. The process of predication will be accomplished by a generalized routine Pred3,which is roughly the semantic counterpart of the match-change process in the syntactic component, with the change procedure "refining" an entry by accumulating of semantic features. Pred3 allows semantic constraints to further restrict the words to be generated.

1.2 Data Structures

The following data structures implement the semantic component of the system.

a. EntrySlots = (Cat, Lab, Instance, Intr, Arg1, Arg2, Locus, Goal, Group);

b. EntryType = array [EntrySlots] of integer;

c. Entry: EntryType;

Entry therefore consists of an ordered array of integers indexing elements in other arrays. The Cat slot indexes a set with information for each syntactic category (described below). The Lab slot indexes a label in the array of lexical category labels. The other slots in the declaration of EntrySlots index either a ternary vector in SemVecs (described below, and indicated by a negative subscript) or another entry in the DiscourseDatabase (described below, and indicated by a positive subscript).

Entry is the molecule of the lexicon and the discourse database.

d. Lexicon: array [ ] of EntryType;

e. LabelType = string[10];

f. LexLabels: array [ ] of LabelType;

g. LexisSpace = array [0 ..a] of Entry-
Type;

h. DatabaseSpace = array [a+1 ..b] of
EntryType;

i. CPSRrefsSpace = array [b+1 ..c] of
Entrytype;

The preceding three type declarations are below used with pointer structures. The three declarations deal with large data sturctures, hence the space allocation is done with pointers.

j. SemVecsSpace = array [-800 ..0] of
TernaryVector;

k. SemFreeType = array [-800 ..0] of
boolean;

```
l.  DatabaseType = record
                E: ^DatabaseSpace;
                Enum:a ..b;(as defined above)
            end;
```

m. DiscourseDatabase: DatabaseType;

As the generator produces new referents, they are added to the database. Pointers are established from the CPSR to old referents in the database as predication proceeds. Thus the text that is generated will obey lexically encoded and database accumulated semantic constraints.

The discourse database concentrates on global focusing, maintaining a list of permanent entities and relations.

```
        n.  CPSRrefsType = record
                          E: ^CPSRrefsSpace;
                          Enum:b ..c; (as above)
                      end;
```

This is used for local focusing, as temporary entities are built up by the sentence.

```
        o.  SemVecOrFree = record
                          V: TernaryVector;
                          Free: boolean;
                      end;

        p.  SemVecs: array of SemVecOrFree;
```

Free indicates whether the corresponding vector (by subscript) is used or free.

q. CPSRrange = (pred, subj, obj, iobj, topic, NPhead, NPmod);

r. CPSR : array [CPSRrange] of integer;

This array indexes database entries. The slots are grammatical roles. Predicative semantics accesses the discourse database entries via the CPSR.

1.3 Some Additional Modules Necessary

a. Function NewSemanticVector creates a new vector in SemVecs in the process of refining or altering an existing semantic vector.

b. Procedure Pred3, given two entries, checks for a match of the vectors and refines the first by the second if a match occurs. Since database entries may point at other entries, Pred3 first ensures that match and refine always get vectors.

Main predicate actions (VTRANS, VINTRANS, PREP and CADJ) and auxiliary actions (TENSE, PROG, PASSIVE, etc) typically map CPSR roles (SUBJ, OBJ, IOBJ, NPHEAD) onto arguments of CPSR[PRED] and call Pred3 to combine CPSR[PRED] with the lexical predicate. Again weights determine the liklihood of categories to be generated.

Pred3 performs the following semantic actions in generating, for example,

"The little girl smiled."

1. In the semantic action associated with the syntactic production NADJ, "little" is generated, and CPSR[NPhead] is refined to "little".

2. In the semantic actions for the syntactic productions NUMBER and N, "girl" is generated,

and CPSR[NPhead] is refined to -PLUR and HUMAN.

3. In the semantic actions for syntactic productions TENSE and VINTRANS, subject-tense agreement is checked, "smiled" is generated, and CPSR[NPhead] is checked as a possible subject of smiled, that is, is CPSR[NPhead] HUMAN?

c. Procedure MakeNewRef produces a new database entry and returns the address (subscript) of that entry.

2. Referential semantics

2.1 Introduction

A Given/New strategy can be implemented as an additional constraint on main predicates, that is, clausal predicates should not only match the arguments, they should say something new about them. This involves the use of assymmetric matching of whole predicate structures, to avoid generating "The little girl smiled, grinning from ear to ear".

2.2 New Data Structures

a. Prons - a record for pronoun maintenance consisting of a Lex field, an array of subscripts for lexical pronouns; Curr and Prev fields, arrays of subscripts for database entries, mentioned in current and previous clause.

At startup, the PRONS list is initialized to contain "you" and "I". Both Prev and Curr are cleared to all zeros. Upon firing NPCLOSE, the database address in CPSR[NPHEAD] is copied into the last unused slot of Curr. At CCLOSE, Prev is cleared, Curr is copied into Prev, and Curr is cleared for another cycle.

b. Names - an array of record pairs of subscripts, one indexing a name string in LexLabels, the other indexing a database entry.

c. SpecialLex - an array of subscripts of lexical entries to be specifically examined by the SemanticActions procedure. The subscripts for SpecialLex are themselves constants (such as INT, DEF, INDEF). For example, Lexicon[SpecialLex[Def]] will index the lexical entry "the".

2.3 Design

At startup, the initial impetus for the type of text to be generated is certainly an area for future consideration. For example, the topic to be discussed may be determined by a query to a database, or the subject of a manual being written or a document being translated.

The design discussed below is that of a random generator. At the beginning of the generation process, the first referent generated will be a new referent; subsequently, the generator will be able to choose whether to generate text pertaining to a new referent or one which exists already.

Once immersed in generating a sentence, the program will decide whether the noun phrase generated will be an old (NPOld) or new (NPNew) referent. Both NPOld and NPNew will have the same condition vectors; choosing will be decided by ordering (Old before New) and randomness of Old (New always succeeds). Their result vectors, however, differ: NPOld (+DEF) will rule out production INDEF, and NPNew (-DEF) will rule out DEF, NAME, PRON, and PRONREFLEX (described below). The semantic action called by NPNew will invoke MakeNewRef, storing the result in CPSR[NPHEAD]. NPOld, on the contrary, will randomly pick a referent from the database.

Order will be important for phrasal productions, and will be enforced by the weights file as described above. In addition, the generator will generate text pertaining to an old referent in preference to producing a new one. The following order is a natural one for the generation process:

a. PRONREFLEX - Reflexives are pronouns specifically used to re-reference the head of a noun clause within the same clause. If CPSR[NPHEAD] is the same referent as CPSR[SUBJ], then the reflexive form (herself, themselves, etc.) should be used. The syntactic condition vector of PRONREFLEX should rule itself out before SUBJ and PRED, so that a reflexive pronoun will not be used as the subject.

b. PRON (Nonlexical) - This action first checks the Current and Previous fields of PRONS to determine if there is a pointer to the database entry of CPSR[NPhead]. If so, AsymmetricMatch3 is called to match the INTR field of CPSR[NPhead] with INTR of an arbitrary pronoun. If the match succeeds, the pronoun form will be used.

c. PRONOBJ (Lexical) - executes if PRON succeeds and syntactic generation has progressed beyond SUBJ and PRED. This requires that an objective form be generated (her, them, etc.).

d. PRONSUBJ (Lexical) - executes if PRON succeeds and syntactic generation is at a point previous to SUBJ and PRED. Then a subjective form is produced (for example, she, they).

e. NAME - if CPSR[NPHEAD] is on the Names list then a proper name will be generated.

f. DEF - will generate "the".

g. INDEF - randomly decide whether to produce a singular or plural phrase; if singular, refine NPHEAD by "a" (-PLUR).

h. NOUN - a referent is in the Current or Previous field of PRONS, but a pronoun or proper name could not be used, then a premodifier is needed, so NOUN will fail. i. NADJ, NVING - will randomly choose a lexical predicate that matches the CPSR[NPHEAD].INTR and refine it. The lexical predicate thus adds descriptive information that will distinguish it from other referents in Prons.Curr or Prons.Prev.

H. Demonstration

Predicative and referential semantics will be implemented as described above to generate sentences. Referential semantic actions are described below for some typical sentences generated (predicative actions are not described):

1st sentence generated:

Did you make the statements?

Referential semantic actions taken:

1. PRON - PRONSUBJ : PRONS contains "you" at initialization; AsymetricMatch3 of CPSR[NPhead] and "you" succeeds; "you" generated.

2. DEF : "the" generated.

3. NOUN : "statements" generated.

2nd sentence generated:

Lisa herself made them.

Referential semantic actions taken:

1. NAME : "Lisa" generated.

2. PRONREFLEX : CPSR[NPhead] = CPSR[SUBJ]; "herself" generated.

3. PRON - PRONOBJ: PRONS.PREV contains reference to "statements"; AsymmetricMatch3 of CPSR[NPhead] and "them" succeeds; "them" generated.

SUMMARY

McCorduck [79] points out that "ours is a history of self-imitation," now in its climax, the chapter which "attempts to reproduce the quintessence of our humanity, our faculty for reason." We should not then expect this to be a trivial effort. There are some weighty issues and problems which we have only begun to study. Natural language generation is an active area of research, boasting a number of interesting systems currently in operation and undoubtedly these will be expanded and others will be developed in the next decade.

The RVG formalism could be a powerful one for natural language generation, and the preceding suggests some avenues for exploiting that capability.

REFERENCES

Appelt, D. "Planning natural language utterances to satisfy multiple goals." Ph.D. thesis, Stanford University, 1981.

Collins, A.M., and Quillian, M.R. "How to Make a Language User". In E. Tulving and W. Donaldson, eds. Organization and Memory. New York: Academic Press, 1972.

Clark,Herbert H., and Marshall, Catherine R. "Definite Reference and Mutual Knowledge". In Joshi, A., et al, Eds. Elements of Discourse Understanding: Proceedings of a Workshop on Computational Aspects of Linguistic Structure and Discourse Setting, Cambridge: Cambridge University Press,1981.

Davey, A. Discourse Production, Edinburg University Press, Edinburgh, 1979.

Derr, Marcia A. and McKeown, Kathleen R. "Using Focus to Generate Complex and Simple Sentences". 1986.

Grosz, Barbara J. "Focusing and Description in Natural Language Dialogues" . In Joshi, A. et al., Eds., Elements of Discourse Understanding: Proceedings of a Workshop on Computational Aspects of Linguistic Structure and Discourse Setting. Cambridge University Press, Cambridge, 1981.

Grosz, Barbara J., Joshi, Aravind K., and Weinstein, Scott. "Providing a Unified Account of Definite Noun Phrases in Discourse". In Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics, 1983.

Kenner, H. "Racter". Byte 11(5), 1986.

Linde, Charlotte. "Focus of Attention and the Choice of Pronouns in Discourse" in Syntax and Semantics, Volume 12: Discourse and Syntax, Talmy Givon, ed., New York: Academic Press, 1979.

Mann, William C., and Moore, James A. "Computer Generation of Multiparagraph English Text". American Journal of Computational Linguistics, 7(1), 1981.

Mann, William C. "Text Generation". American Journal of Computational Linguistics, 8(2), 1982.

Mann, William C. "An Overview of the Penman Text Generation System, USC/ISI Technical Report RR-83-114, 1983.

Mann, William C. "Discourse Structures for Text Generation". In Proceedings of the 22nd Annual Meeting of the Association for Computational Linguistics, 1984.

Mauldin, Michael L. "Semantic Rule Bassed Text Generation". In Proceedings of the 22nd Annual Meeting of the Association for Computational Linguistics, 1984.

McCorduck, Pamela. Machines Who Think. San Francisco: W. H. Freeman & Co., 1979.

McDonald, David D. "Natural Language Generation as a Computational Problem: an Introduction" from "Natural Language Production as a Process of Decision-Making Under Constraints. Ph.D. thesis, Massachusetts Institute of Technology. 1980.

McDonald, David D. and Vaughn, M.M. "A Model of Revision in Natural Language Generation". In Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics, 1986.

Rosenschein, Stanley J. "Abstract theories of discourse and the formal specification of programs that converse". In Joshi et al, Eds., Elements of Discourse Understanding: Proceedings of a Workshop on Computational Aspects of Linguistic Structure and Discourse Setting, Cambridge: Cambridge University Press, 1981.

Sidner, Candace L. "Focusing for Interpretation of Pronouns". American Journal of Computational Linguistics, Volume 7, Number 4, 1981.

Smith, E.E., Shoben, E.J., and Rips, L.J. "Comparison Processes in Semantic Memory" Psychological Review (81), 1974.

Woods, William A. "Procedural Semantics". In Joshi, A., et al, Eds., Elements of Discourse Understanding: Proceedings of a Workshop on Computational Aspects of Linguistic Structure and Discourse Setting, 1981.

VITA

Marie Bettinger Wilde is Assistant Professor of Mathematics and Computer Science at Cedar Crest College, Allentown, Pennsylvania, and holds bachelor's and master's degrees in mathematics. She resides near Bethlehem with her husband and two daughters.